01.09.2004

日本国特許庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application: ２００３年　８月２９日

出願番号
Application Number: 特願２００３－３０６３５７

[ST. 10/C]: [ＪＰ２００３－３０６３５７]

出願人
Applicant(s): アイピーフレックス株式会社

２００４年１０月　７日

特許庁長官
Commissioner,
Japan Patent Office

小川洋

出証番号　出証特２００４－３０９０２７９

【書類名】　特許願
【整理番号】　030163P502
【あて先】　特許庁長官殿
【国際特許分類】　H03K 19/00
【発明者】
　【住所又は居所】　東京都品川区上大崎二丁目２７番１号　アイピーフレックス株式会社内
　【氏名】　佐藤　友美
【特許出願人】
　【識別番号】　500238789
　【氏名又は名称】　アイピーフレックス株式会社
【代理人】
　【識別番号】　100102934
　【弁理士】
　【氏名又は名称】　今井　彰
【手数料の表示】
　【予納台帳番号】　050728
　【納付金額】　21,000円
【提出物件の目録】
　【物件名】　特許請求の範囲　1
　【物件名】　明細書　1
　【物件名】　図面　1
　【物件名】　要約書　1

【書類名】特許請求の範囲
【請求項１】
回路を動的に再構成可能な論理回路領域を有するデータ処理装置の制御方法であって、
ある機能単位として設計されるハードウェアモジュールを１または複数に分割した分割回路を前記論理回路領域の一部にマッピングする分割回路情報と、前記分割回路に接するインタフェース回路を前記論理回路領域にマッピングするインタフェース回路情報と、前記インタフェース回路において実現する境界条件とを備えたアーキテクチャコードを取得する工程と、
前記アーキテクチャコードの前記分割回路情報およびインタフェース回路情報により、前記論理回路領域に前記分割回路と、その分割回路に接する前記インタフェース回路とをマッピングする工程と、
前記アーキテクチャコードの前記境界条件に基づき前記インタフェース回路を制御する動作工程とを有するデータ処理装置の制御方法。
【請求項２】
請求項１において、前記マッピングする工程では、前記分割回路とインタフェース回路とを、前記論理回路領域の利用可能ないずれかの領域にマッピングする、データ処理装置の制御方法。
【請求項３】
請求項１において、前記マッピングする工程では、隣接する分割回路に接する境界の前記インタフェース回路情報および境界条件が一致または対応する場合は、前記インタフェース回路を経ずに前記隣接する分割回路に接するように前記分割回路をマッピングする、データ処理装置の制御方法。
【請求項４】
請求項１において、前記動作工程では、前記論理回路領域に現在および／または過去にマッピングされた他の分割回路の他のインタフェース回路の状態が前記境界条件に基づき前記インタフェース回路の制御に反映される、データ処理装置の制御方法。
【請求項５】
請求項１において、前記取得する工程では、当該データ処理装置に対する要求、マッピングされた前記分割回路の実行状況、前記論理回路領域の利用可能状況を含めた動作環境情報に基づき、取得する前記アーキテクチャコードを選択する、データ処理装置の制御方法。
【請求項６】
請求項１において、前記論理回路領域は複数の再構成可能な複数のエレメントと、所定の数の前記複数のエレメントにより構成される回路ブロックを複数備えており、前記アーキテクチャコードは、前記回路ブロックの単位の前記分割回路情報を含む、データ処理装置の制御方法。
【請求項７】
回路を動的に再構成可能な論理回路領域と、
ある機能単位として設計されるハードウェアモジュールを１または複数に分割した分割回路を前記論理回路領域の一部にマッピングする分割回路情報と、前記分割回路に接するインタフェース回路を前記論理回路領域にマッピングするインタフェース回路情報と、前記インタフェース回路において実現する境界条件とを備えたアーキテクチャコードを取得するロードユニットと、
前記アーキテクチャコードの前記分割回路情報およびインタフェース回路情報により、前記論理回路領域に前記分割回路と、その分割回路に接する前記インタフェース回路とをマッピングするマッピングユニットと、
前記アーキテクチャコードの前記境界条件にしたがって前記インタフェース回路を制御する動作制御ユニットとを有するデータ処理装置。
【請求項８】
請求項７において、前記マッピングユニットは、前記分割回路とインタフェース回路と

を、前記論理回路領域の利用可能ないずれかの領域にマッピングする、データ処理装置。
【請求項９】
請求項７において、前記マッピングユニットは、隣接する分割回路に接する境界の前記インタフェース回路情報および境界条件が一致または対応する場合は、前記インタフェース回路を経ずに前記隣接する分割回路に接するように前記分割回路をマッピングする、データ処理装置。
【請求項１０】
請求項７において、前記動作制御ユニットは、前記論理回路領域に現在および／または過去にマッピングされた他の分割回路の他のインタフェース回路の状態を前記境界条件に基づき前記インタフェース回路の制御に反映する、データ処理装置。
【請求項１１】
請求項７において、前記論理回路領域に現在および／または過去にマッピングされた分割回路のインタフェース回路の状態を記憶する境界情報メモリを有するデータ処理装置。
【請求項１２】
請求項７において、前記ロードユニットは、当該データ処理装置に対する要求、マッピングされた前記分割回路の実行状況、前記論理回路領域の利用可能状況を含めた動作環境情報に基づき、複数のアーキテクチャコードを備えたアーキテクチャライブラリから所望の前記アーキテクチャコードを取得する、データ処理装置。
【請求項１３】
請求項７において、複数のハードウェアモジュールを構成する複数のアーキテクチャコードを備えたアーキテクチャライブラリを有するデータ処理装置。
【請求項１４】
請求項７において、前記論理回路領域は複数の再構成可能な複数のエレメントと、所定の数の前記複数のエレメントにより構成される回路ブロックを複数備えており、前記アーキテクチャコードは、前記回路ブロックの単位の前記分割回路情報を含む、データ処理装置。
【請求項１５】
請求項１４において、前記エレメントは、４系統の入力と、４系統の出力と、前記４系統の入力から任意の入力データを選択する入力インタフェースと、この入力インタフェースにより選択された入力データを論理演算して出力データを出力する演算コアであって、その論理演算を変更可能な演算コアと、前記４系統の入力と前記出力データとを任意に選択して前記４系統の出力へ接続可能な出力インタフェースとを備えている、データ処理装置。
【請求項１６】
請求項１５において、前記演算コアは、論理演算を指示する多ビットのファンクションコードが入力され、前記入力データにより前記出力データを選択するセレクタを備えている、データ処理装置。
【請求項１７】
請求項１５において、前記演算コアは、前記４系統の入力のいずれかのデータ、または、前記出力データをラッチするレジスタを備えている、データ処理装置。
【請求項１８】
回路を動的に再構成可能な論理回路領域を有するデータ処理装置を制御するためのアーキテクチャコードであって、ある機能単位として設計されるハードウェアモジュールを１または複数に分割した分割回路を前記論理回路領域の一部にマッピングする分割回路情報と、前記分割回路に接するインタフェース回路を前記論理回路領域にマッピングするインタフェース回路情報と、前記インタフェース回路において実現する境界条件とを備えたアーキテクチャコードを記録していることを特徴とする記録媒体。
【請求項１９】
回路を動的に再構成可能な論理回路領域を有するデータ処理装置を制御するためのアーキテクチャコードであって、ある機能単位として設計されるハードウェアモジュールを１

または複数に分割した分割回路を前記論理回路領域の一部にマッピングする分割回路情報と、前記分割回路に接するインタフェース回路を前記論理回路領域にマッピングするインタフェース回路情報と、前記インタフェース回路において実現する境界条件とを備えたアーキテクチャコードの生成方法であって、

前記ハードウェアモジュールのオリジナルのネットリストを１または複数に分割して配置および配線問題を解決して前記分割回路情報を生成する工程と、

前記オリジナルのネットリストの前記分割回路の境界を形成する情報から前記インタフェース回路情報を生成する工程と、

前記オリジナルのネットリストを前記分割回路の集合に変換し、それらの分割回路間の配置・配線問題を解決し、前記インタフェース回路における境界条件を生成する工程とを有する、アーキテクチャコードの生成方法。

【請求項２０】

回路を動的に再構成可能な論理回路領域を有するデータ処理装置であって、

前記論理回路領域は複数の再構成可能な複数のエレメントを備え、

前記エレメントは、入力データを論理演算して出力データを出力する演算コアを備えており、前記演算コアは、論理演算を指示する多ビットのファンクションコードが入力され、前記入力データにより前記出力データを選択するセレクタを備えている、データ処理装置。

【請求項２１】

請求項２０において、前記エレメントは、４系統の入力と、４系統の出力と、前記４系統の入力から任意の前記入力データを選択する入力インタフェースと、前記４系統の入力と前記出力データとを任意に選択して前記４系統の出力へ接続可能な出力インタフェースとを備えている、データ処理装置。

【請求項２２】

請求項２０において、前記演算コアは、前記４系統の入力のいずれかのデータ、または、前記出力データをラッチするレジスタを備えている、データ処理装置。

【請求項２３】

請求項２０において、所定の数の前記複数のエレメントにより構成される回路ブロックを複数備えている、データ処理装置。

【書類名】明細書
【発明の名称】データ処理装置
【技術分野】
【０００１】
本発明は、再構成可能な論理回路領域を有するデータ処理装置に関するものである。
【背景技術】
【０００２】
回路を再構成できるプログラマブルデバイスとして、ＦＰＧＡ（Field Programmable Gate Array）、ＰＬＤ（Programmable Logic Device）、ＰＬＡ（Programmable Logic Array）と称されるデバイスが知られている。これらのプログラマブルデバイスの基本的な構成は、論理セルあるいは論理ユニットと称されるユニットが格子状に配置され、それを取り巻くように配線群が配置されたものであり、コンテキスト情報あるいはコンフィグレーション情報と呼ばれる情報によって、論理セルの機能や配線の接続を変更できるようになっている。
【特許文献１】特開２０００－４０７４５号公報
【発明の開示】
【発明が解決しようとする課題】
【０００３】
特開２０００－４０７４５号公報には、ＦＰＧＡに論理回路の異なる部分を実装する技術の１つとして、論理回路を特徴付ける初期ネットリストを多くのページへと区分し、ＦＰＧＡにこれらのページの１つの回路を実装することが記載されている。これにより、ＦＰＧＡの物理的容量よりもはるかに大きな回路の実装を可能にしようとしている。
【０００４】
しかしながら、現在、マルチメディアデバイス、モバイルデバイス、デジタルデバイスなどに搭載され、それらのデバイスのデータ処理の多くを行っているシステムＬＳＩは、１つのチップの上に、特定の機能を実現するための回路単位（多くのケースでは、ハードウェアモジュールあるいはＩＰ（Intellectual Property）、ライブラリと称される）が複数搭載され、それらのハードウェアモジュールが並列して処理を行っている。したがって、ＦＰＧＡに、単に１つの回路を分解して実装したとしても、回路を再構成可能なデバイスの有効性が大きく広がることにはならない。
【０００５】
これに対し、本発明においては、アプリケーションを実行するために、ハードウェア空間を動的に最適化する技術を提供する。そして、コンパイラ翻訳による命令セットプログラムのような従来のソフトウェア情報だけでは無く、ハードウェア回路そのものを直接実行可能とするアーキテクチャを備えたＬＳＩを本発明においては提供する。
【０００６】
従来のシステムＬＳＩに代表される回路デバイスの技術においては、ハードウェア回路は、特定のハードウェアモジュールやＩＰ、ライブラリと呼ばれる単位で搭載され、専用化されたＬＳＩによりデータが処理される。これに対し、汎用的な回路あるいはアーキテクチャでハードウェア回路を実行する技術がある。その１つは、シミュレータのように命令プログラムを１つ１つ実行して、その回路自体を実行しているかのように割り当てる手法である。これは、本来ハードウェア回路の持つ並列性を命令単位の実行に変えて処理を行うため、複数のＣＰＵを使用したとしても、実際のハードウェア回路と比較して、実行する回路規模にもよるが、普通３桁～５桁以上の実行時間が必要とされる。また、リアルタイム性に決定的に欠けるので、実際に複雑な回路をシミュレーションしようとすると膨大な検証時間を必要とする。このため、膨大なゲート数を有する近年の専用ＬＳＩに代わり処理を実行することが不可能なだけではなく、複雑な専用ＬＳＩの機能を検証するにも不向きな状況になりつつある。
【０００７】
リアルタイム性の問題に対して、ハードウェア・アクセラレーションと呼ばれる手法が

ある。これは、最初の段階では、ＣＰＵやＤＳＰを複数並べて並列実行させ、その一つ一つに小規模な回路を割り当てて、全体としてはシミュレーション時間を短時間で行うというアイデアである。ＦＰＧＡやＰＬＤが実用化されたことにより、シミュレーション対象の回路を直接これに割り当てるやり方が主流となり、大規模な集積回路やハードウェアのエミュレーションが、リアルタイムに極めて近い時間で実行できるようになりつつある。

【０００８】

しかしながら、ＦＰＧＡの内部セル構造は、ハードウェア回路の実現をある一定の時間を掛けて変更するようなアーキテクチャ構造となっており、ＣＰＵやＤＳＰ等のデータパス系を有するハードウェアの実装にはあまり向いていない。実際に、実装しようとすると、処理性能（動作周波数）・ゲート効率・消費電力の何れでも、専用に設計されたＬＳＩと競争できない。アプリケーションを実行するためのターゲット回路にもよるが、ＦＰＧＡ・ＰＬＤの場合、実装対象となるターゲット回路の回路規模がＦＰＧＡ・ＰＬＤの集積度より大きい場合、原則として実装は不可能となる。ターゲット回路を分割して実装する事も考えられるが、その場合、チップが複数になりピン数の制限から、更に性能・コスト・消費電力とも不利になる。

【０００９】

上述したように１つのＦＰＧＡに分割した回路を実装することも検討されているが、ピン数の制限や、分割した回路同士の境界情報の伝播など、回路を分割する際に発生する問題が多数あり、それらをハードウェア設計の段階ですべて解決しようとすると、専用ＬＳＩを設計および開発する従来の技術に対するメリットは失われてしまう。

【００１０】

さらに、一般的に、ＦＰＧＡやＰＬＤは、目標のハードウェア回路の数倍～数十倍ものハードウェアが必要となり、チップ・コスト、目標性能、および消費電力の３つの点で、専用ＬＳＩには及ばない。

【００１１】

一方、専用ＬＳＩにも多くの問題がある。従来の専用ＬＳＩの場合、ＬＳＩの設計段階で正確な性能目標や機能仕様が無いと設計が収束しない。例えば、アプリケーションの実行状況によっては、機能と性能の動的トレード・オフが可能なことが多いが、設計段階で必要とされる性能を実現可能なだけのハードウェア領域や性能を保証出来る動作周波数を確定させる必要がある。つまり、機能や性能の動的トレード・オフがアプリケーション上可能な場合でも、ピーク性能要求や単体レベルの機能毎の性能保証を行った上で、ＬＳＩ設計を行う必要がある。従って、機能と性能についての要求が決定的であり、高性能および多機能を狙うシステムＬＳＩは、コスト的には最悪ケースの積み上げとなり、製造コスト、チップ面積、消費電力などが悪化する一方になる。

【００１２】

もう少し具体的な例としては、ロボットのようなアプリケーションの場合、視覚情報や聴覚情報を処理しているときは、他の機能（歩行機能・言語処理・嗅覚処理等）を大幅に弱めて良い場合が多い。しかしながら、従来のシステムＬＳＩでは、すべての機能を実現するすべての回路を同じようにシステムＬＳＩに実装しており、単にその処理結果を使用しないか、あるいは、スタンバイ状態で処理能力を低下させているに過ぎない。

【００１３】

これに対し、回路構成を動的に再構成できるのであれば、その再構成可能な論理回路により構成されるハードウェア空間を動的に最適化し、使用しない、あるいはスタンバイ状態になる機能へのハードウェア資源の割り当てを大幅に絞って、本来集中すべき視覚情報処理や聴覚情報処理にハードウェア資源を集中的に割り当てることができる。すなわち、本発明によれば、従来の専用ＬＳＩのように、ハードウェア回路（ターゲット回路）をすべて実装する必要がないので、少ないハードウェア資源で最大の実行効率を得ることができる。

【００１４】

本発明における動的最適化技術は、論理回路により構成される実ハードウェア空間のア

レンジを動的に最適化することを言い、実ハードウェア空間をその都度刷新するだけではなく、実ハードウェア空間の部分的なアレンジを動的に最適化することも含む概念である。したがって、現在使用していない機能へのハードウェア資源の割り当てを無くすだけではなく、ハードウェア資源の割り当てを絞り、スタンバイ中の機能のリアルタイム応答性を犠牲にすることなく、使用中の機能に対するハードウェア資源の割り当てを増加させることができるものである。

【００１５】

また、本発明における動的最適化技術は、使用中であっても緊急性を要する機能に対してはハードウェア資源の割り当てを増加し、緊急性を要しない機能に対してはハードウェア資源の割り当てを減らしたり、ハードウェア資源の割り当てを一時的に無くすことができるものである。緊急性とは、処理速度、優先順位などを含む概念であり、データ処理装置に対する要求の重要なものの１つであるが、その他に、ハードウェア資源の割り当てを左右する、データ処理装置に対する要求としては、並列処理するジョブの増減、割り込みの有無など様々なものが考えられる。本発明における動的最適化技術は、これらのデータ処理装置に対する要求に応じてハードウェア資源の割り当て、すなわち、実ハードウェア空間の構成を動的に最適化する。

【課題を解決するための手段】

【００１６】

実ハードウェア空間を動的に最適化する１つの方法は、データ処理装置が遭遇するすべての場面を想定し、それに対して最適な実ハードウェア空間のアレンジを予め決定し、コンテキスト情報（あるいはコンフィグレーション情報）として用意し、その都度、ロードする方法である。この方法は、実ハードウェア空間で生ずるタイミング収束などの問題を予め解決できるので、データ処理装置の性能を確保する点では望ましいかもしれない。しかしながら、すべてのシナリオにおいて遭遇する場面を想定することは不可能であり、限られた場面を対象としてある程度最適化し、その他の場面では中庸な性能が得られるような汎用的な解を得ようとすれば、実ハードウェア空間を動的に最適化する効果は薄れてしまう。

【００１７】

他の方法の１つは、機能単位として設計されるハードウェアモジュール（ＩＰまたはライブラリ）を、論理ゲートとそれらの接続状態を示しただけのネットリストの状態で用意し、その機能が必要となったときに、そのネットリストの一部あるいは全体を実ハードウェア空間の空いた空間にマッピングするために動的に配置および配線する方法である。この方法は、実ハードウェア空間の瞬間瞬間の状況に合わせて回路をフレキシブルに、動的に配置できるので、ハードウェア空間を最も動的に最適化できる方法であると考えられる。しかしながら、ＬＳＩの設計および開発段階でも膨大な時間を要するネットリストに基づく配置および配線する処理を瞬間瞬間で繰り返す必要があり、瞬間瞬間の実ハードウェア空間の実情とその他の要素を加味して実際にネットリストに基づいて配置・配線の諸問題を解決してマッピングすることは不可能である。ほとんどクロック単位あるいはサイクル単位で高速にタイミング収束を含めた問題を解決できるハードウェアが提供できたとしても、そのようなハードウェア資源を別途用意することは経済的でもないし、そのようなハードウェアの登場を待つのでは、ハードウェア空間の動的最適化の実現が難しくなるだけである。

【００１８】

ネットリストの状態から、ある程度の単位で配置・配線を行って、それらの回路単位を実ハードウェア空間の空いた空間に配置して、それらの回路単位を接続する配置・配線問題に縮小することにより、瞬間瞬間での配置・配線問題を解決する時間を短縮できる可能性がある。しかしながら、配置・配線するときの実ハードウェア空間の状況は刻々と変動するので、それに対して常に瞬間瞬間で配置・配線問題を動的に解決することは容易ではないであろうし、可能であったとしてもハードウェア資源と電力をそのために常に費やすことには変わりなく、高性能高機能・低チップコスト・低消費電力のデータ処理装置を提

供するという課題を解決することができない。

【００１９】

そこで、本発明においては、ある機能単位として設計されるハードウェアモジュールを１または複数に分割した分割回路を論理回路領域の一部にマッピングする分割回路情報と、分割回路に接するインタフェース回路を論理回路領域にマッピングするインタフェース回路情報と、インタフェース回路において実現する境界条件とを備えたアーキテクチャコードを提供する。回路を動的に再構成可能な論理回路領域を有するデータ処理装置に対する本発明の制御方法は、アーキテクチャコードを取得する工程と、アーキテクチャコードの分割回路情報およびインタフェース回路情報により、論理回路領域に分割回路と、その分割回路に接するインタフェース回路とをマッピングする工程と、アーキテクチャコードの境界条件に基づきインタフェース回路を制御する動作工程とを有する。

【００２０】

また、本発明のデータ処理装置は、回路を動的に再構成可能な論理回路領域と、アーキテクチャコードを取得するロードユニットと、アーキテクチャコードの分割回路情報およびインタフェース回路情報により、論理回路領域に分割回路と、その分割回路に接するインタフェース回路とをマッピングするマッピングユニットと、アーキテクチャコードの境界条件にしたがってインタフェース回路を制御する動作制御ユニットとを有する。ロードユニットは、アーキテクチャコードをフェッチする場合はフェッチユニットであり、ネットワークやメモリからダウンロードする場合はダウンロードユニットになる。ロードした回路情報によりハードウェアを再構成するマッピングを含めてロードと呼ばれることもあるが、本明細書においては、アーキテクチャコードを取得するまでをロードするステップと呼ぶことにする。ロードユニットにおいてコードを取得するプロセスには、フェッチ、ダウンロード、ゲット、リードなど様々な命令を割り当てることができ、コミュニケーションシステムによりアーキテクチャコードをロードすることも可能である。

【００２１】

本発明においては、ハードウェアモジュールのオリジナルのネットリストからある程度の単位で配置・配線問題が解決され、そのまま論理回路領域の一部にマッピング可能な分割回路情報を生成する。次に、オリジナルのネットリストの分割回路の境界を形成する情報からインタフェース回路情報を生成し、さらに、オリジナルのネットリストを分割回路の集合に変換して、それらの分割回路の間の配置・配線問題を解決し、インタフェース回路における境界条件を生成する。したがって、分割回路間の配置・配線問題は、インタフェース回路における境界条件として、アーキテクチャコードの生成段階で解決されている。このため、分割回路を実ハードウェア空間の空いた空間に配置するときには、ロードユニットにより、適切なアーキテクチャコードを取得し、マッピングユニットにより分割回路をマッピングすると共に、その周囲にインタフェース回路をマッピングし、実行制御ユニットにより、インタフェース回路を境界条件に基づいて制御するだけで、実ハードウェア空間に分割回路を動的に配置し、分割回路を実行することができる。したがって、実ハードウェア空間の瞬間瞬間の状態により、所望の分割回路とインタフェース回路をアレンジするだけで分割回路を実行することができ、瞬間瞬間で分割回路間の配置および配線問題を解決しなければならないという問題の発生を未然に防止できる。

【００２２】

本発明においては、実ハードウェア空間にマッピングされた１つまたは複数の分割回路は、その状態で実行されるが、分割回路の境界はインタフェース回路を介して仮想的にはハードウェアモジュールを構成する多数の分割回路に接続されており、分割回路の境界は多数の分割回路が接続された仮想ハードウェア空間の状態で制御される。したがって、マッピングする際は、分割回路とインタフェース回路とを、論理回路領域の利用可能ないずれの領域にもマッピングすることができる。さらに、隣接する分割回路に接する境界のインタフェース回路情報および境界条件が一致する、または対応している場合は、仮想ハードウェア空間において隣接している分割回路を意味するので、インタフェース回路を経ずに隣接する分割回路同士を直に接するように分割回路をマッピングすることができる。す

なわち、実ハードウェア空間にマッピングする実際の分割回路の大きさを実ハードウェア空間の状態に合わせて自由に変えることができる。また、複数の分割回路を実ハードウェア空間に分散してマッピングすることも、集中してマッピングすることも可能であり、実ハードウェア空間を極めてフレキシブルに使用することができる。

【００２３】

論理回路領域に現在および／または過去に、時間および／または空間的に分散配置された他の分割回路とは、マッピングされた他の分割回路の他のインタフェース回路の状態を、動作制御ユニットにより、境界条件に基づき、実行するターゲットの分割回路のインタフェース回路の制御に反映することにより、仮想ハードウェア空間では無理なく接続することができる。このため、論理回路領域に現在および／または過去にマッピングされた分割回路のインタフェース回路の状態を記憶する境界情報メモリを設けておくことが望ましい。

【００２４】

本発明のアーキテクチャコードは、様々な利用方法がある。実ハードウェア空間にマッピングするアーキテクチャコードをプログラムの命令セットのようにシーケンシャルにトレースできる状態で提供することにより、アーキテクチャコードによりデータ処理装置を制御できる。アーキテクチャコードは、記録媒体に記録して提供することも可能であるし、ネットワークなどの通信手段を介して提供することも可能であり、ハードウェアの構成を遠隔操作により変えることも可能である。

【００２５】

また、従来のシステムＬＳＩに代わる使用方法としては、ロードユニットにより、データ処理装置に対する要求（緊急性や、新たなジョブの開始あるいは並列処理状態の変化、割り込みの有無など）、マッピングされた分割回路の実行状況、論理回路領域の利用可能状況を含めた動作環境情報に基づき、複数のアーキテクチャコードを備えたアーキテクチャライブラリから所望のアーキテクチャコードを取得し、アプリケーションの実行状況によりデータ処理装置のハードウェアを動的に最適な構成にすることができる。最適化の指針は、動作環境情報に基づき決めることができ、それにはロードユニット、マッピングユニット、実行制御ユニットの１つまたは複数が寄与する。実ハードウェア空間の空いた空間に新たな分割回路をマッピングしたり、使用済みの分割回路を消去したり、緊急性を有するハードウェアモジュールを優先的にマッピングするために、他のハードウェアモジュールの分割回路を一時的に退避したり、他のハードウェアモジュールに割り当てられたハードウェア資源の割合を一時的に縮小したり、実ハードウェア空間の利用方法に制限はない。

【００２６】

また、ハードウェア回路をアーキテクチャコード化することにより、データ処理装置の利用価値は飛躍的に増大する。限られた利用方法としては、アーキテクチャライブラリをデータ処理装置に実装することができる。一方、ネットワークなどのデータ処理装置の外側に対する通信を介してアーキテクチャコードを取得することも可能であり、分割回路をマッピングできる程度のハードウェア空間があれば、膨大なハードウェア資産を自由に利用することができる。たとえばインターネット上に存在する多種多様なハードウェア資産を手元の携帯端末のＬＳＩにマッピングして利用することが可能となる。

【００２７】

分割回路のサイズはフレキシブルであり、マッピング対象の論理回路領域にインタフェース回路も含めてマッピングできるサイズよりも小さければ良い。分割回路のサイズは小さい方が実ハードウェア空間の最適化の効率は高い。しかしながら、１つのハードウェアモジュールに対する用意されるアーキテクチャコードの量が多くなる。マッピングする際は、本発明においては、ハードウェアスペースさえあれば複数の分割回路をまとめてマッピングすることが可能なので、分割回路のサイズによりマッピングを繰り返す数が膨大になって処理時間が増大するような心配は少ない。多くの再構成可能なアーキテクチャは、複数の再構成可能な複数のエレメントと、所定の数の複数のエレメントにより構成される

回路ブロックを複数備えているので、アーキテクチャコードは、回路ブロックの単位の分割回路情報を含むことが望ましい。

【００２８】

本発明のアーキテクチャコードは、回路を動的に再構成可能な論理回路領域を有するすべてのデータ処理装置に対して適用できる。しかしながら、ＦＰＧＡのように回路の構成をルックアップテーブル（ＬＵＴ）に記憶するハードウェアであると、ＬＵＴを変更するために数クロックを有し、実行速度の遅れが目立つ可能性がある。したがって、本発明においては、再構成に要する時間が非常に短い複数のエレメントを備えた論理回路領域を有するデータ処理装置を提供する。

【００２９】

本発明のエレメントは、入力データを論理演算して出力データを出力する演算コアを備えており、演算コアは、論理演算を指示する多ビットのファンクションコードが入力され、入力データにより出力データを選択するセレクタを備えている。演算コアに入力されているファンクションコードを供給するだけで論理を変更できるので、ＬＵＴを書き換える必要がなく高速に論理を交換できる。

【００３０】

さらに、エレメントは、４系統の入力と、４系統の出力と、４系統の入力から任意の入力データを選択する入力インタフェースと、４系統の入力と出力データとを任意に選択して４系統の出力へ接続可能な出力インタフェースとを備えていることが望ましい。４方向のいずれからもデータを入力でき、４方向のいずれに対してもデータを出力でき、さらに、このエレメントは論理演算しない単なる接続切り換えエレメントとしても機能する。さらに、演算コアに、４系統の入力のいずれかのデータ、または、出力データをラッチするレジスタを設けることが望ましい。レジスタを使用しなければ、デコーダなどのクロック依存性のない、あるいは少ない処理を実行するのに適した回路を構成でき、レジスタを使用すれば、ステートマシンなどのクロック依存性の高い処理に適した回路を構成できる。

【発明の効果】

【００３１】

本発明によれば、数クロック・サイクル単位で動的最適化が可能となるので、トレード・オフの自由度が非常に大きく、高性能高機能・低チップコスト・低消費電力という相反する要求を同時に高いレベルで実現することができる。

【００３２】

したがって、リコンフィギャブル・テクノロジーの有効性は飛躍的に増大し、実装効率が向上し、専用ＬＳＩと比較して内部の稼働効率を格段に向上させ、チップ・コスト、性能及び消費電力の３つの点で有利な解決手段を提供することができる。また、プログラマブルなハードウェアである特徴は最大限に活かされるので、従来のＬＳＩ開発手法では、物理デザインと機能検証・品質保証の為に数ヶ月以上の時間を必要としたが、原理的にこれを必要としないアーキテクチャを提供できる。

【００３３】

すなわち、本発明においては、実現可能なハードウェア空間の動的最適化技術を提供しており、従来のハードウェア回路情報及び命令プログラムのようなハードウェア回路を制御するソフトウェア情報を１つの統一されたアーキテクチャコードと呼ばれる体系で符号化して、ハードウェアの実行サイクル上、アプリケーション要求（アーキテクチャ）の拘束条件下で、その瞬間瞬間に有効となるハードウェア・リソースと要求される処理性能をトレードオフし、動的に最適化を行う事で、特定アプリケーション用に設計された専用ＬＳＩ対しても、チップ・コスト、性能要求・消費電力の３つの面で優位性を示すことが可能となっている。

【００３４】

ハードウェア空間の動的最適化を実現する為のアーキテクチャの基本要素技術は、ハードウェア回路の時分割実行可能とする回路分割実行と継続実行技術、複数のハードウェア回路間のチャネル接続技術、動的ハードウェア回路生成技術・縮小技術・消去交換技術、

ハードウェア回路情報のコンパクト化技術と回路情報の高速移動技術、アプリケーション要求とリソース間トレード・オフ・ソフトウェア技術、ハードウェア回路やソフトウェア情報の高速エミュレーション技術を挙げる事が出来る。

【００３５】

また、本発明のアーキテクチャコードは、分割回路情報、インタフェース回路情報、境界条件を含むものであるが、さらには、アーキテクチャコードは、ハードウェア回路情報とソフトウェア情報の２つに大きく分類される。ハードウェア回路情報は、使用可能なハードウェア資源を１００％とした場合の各回路のモジュール相関情報（静的トポロジー情報と動的モジュール実行情報）・階層構造・優先順位・例外処理条件・動的トレード・オフ条件等を含むことができる。ソフトウェア情報は、従来の命令プログラムやベクターテーブル、アプリケーションによっては、画像等のデータ情報を含むことができる。一般的な意味では、ハードウェア回路の機能やタイミング制御を補助的にサポートする情報全体を含むことができる。

【００３６】

マイクロ・プロセッサーの場合、外部割り込み制御部や例外処理部のような特定条件でしか必要とされない回路やデコーダやデータ・パスのように比較的高い頻度で実行されるような回路が存在する。ハードウェア空間とは、階層的に構成されるハードウェア回路をマイクロ・プロセッサーという機能面から分析し直して、動的実行を想定した動的最適化が可能なように再構成した情報の集合体を指す。一般的なプログラム空間とは異なり、並列実行を想定した複数のハードウェア回路情報から構成され、回路交換の余裕が無いものに関してはＬＳＩ内部のメモリに直接実装される。逆に、回路実行や起動余裕のあるものは、外部メモリから内部メモリにロードされ実行される。

【００３７】

本発明のデータ処理装置であるアーキテクチャＬＳＩは、ロードユニットおよびマッピングユニットとしての機能を備え、アーキテクチャコードをハードウェア上で翻訳してハードウェアの初期化や分割実行する為の外部・内部の高速ローディング制御ユニット（ＲＬＣ）、実行制御ユニットとしての機能を備え、高速論理回路交換動作の制御と階層的情報伝達（伝送）を行う高速論理通信マスタ（ＲＴＭ）、論理回路領域となる、各種ハードウェア回路（テスト回路含む）を直接分割実行する高速論理回路交換エレメント（ＲＸＥ）群から構成することができる。

【発明を実施するための最良の形態】

【００３８】

図１に、本発明のデータ処理装置の一例を示してある。このデータ処理装置１は、回路を動的に再構成可能な論理回路領域（ＲＣ領域、リコンフィグラブル領域）１０と、幾つかのハードウェアモジュールのアーキテクチャコード２０を記録したアーキテクチャライブラリ２と、アーキテクチャコード２０を再構成可能なハードウェア１０の上で翻訳してハードウェアの初期化や分割実行する高速ローディング制御ユニット（ＲＬＣ）１１と、高速論理回路交換動作の制御と階層的情報伝達（伝送）を行う高速論理通信マスタ（ＲＴＭ）１２とを備えている。ＲＬＣ１１は、ライブラリ２からアーキテクチャコード２０を取得（フェッチあるいはダウンロード）するロードユニット１３と、アーキテクチャコード２０の分割回路情報およびインタフェース回路情報により、ＲＣ領域１０に分割回路１９と、その分割回路１９に接するインタフェース回路１８とをマッピングするマッピングユニット１４としての機能を備えている。ＲＴＭ１２は、アーキテクチャコードの境界条件にしたがってインタフェース回路１８を制御する動作制御ユニットとしての機能を備えている。また、ＲＴＭ１２は、ＲＣ領域１０に現在および／または過去にマッピングされた分割回路１９のインタフェース回路１８の状態を、必要に応じて境界情報メモリ１５に記憶し、分割回路１９の間の情報伝達を行う。

【００３９】

さらに、データ処理装置１は、アプリケーションを実行するプログラム４が記憶されたＲＡＭ５と、プログラム４にしたがってデータ処理装置１のハードウェア資源を用いて処

理を実行するＲＩＳＣプロセッサ６と、割り込み信号を受信する割り込み制御ユニット７と、データ処理装置１の各ハードウェア資源にクロック信号を供給するクロック発生源８と、外部メモリに対してデータの入出力を制御するデータ入出力インタフェース９とを備えている。コードＲＡＭ５は、ＲＣ領域１０からもアクセスできるようになっている。

【００４０】

本例のデータ処理装置１においては、ＲＣ領域１０に、ＣＰＵあるいはＤＳＰとしての機能をマッピングすることが可能なので、ＲＣ領域１０でコードＲＡＭ５に記憶されたプログラム４を実行することが可能である。したがって、ＲＩＳＣプロセッサ６の機能を、ＲＣ領域１０を用いて実現することにより、ＲＣ領域１０にマッピングされる回路によりＲＣ領域１０の制御も含めたデータ処理装置１の制御を行うことも可能である。この場合、データ処理装置１において構成が固定されたハードウェアで実現される機能は、初期設定を行う機能と、ＲＴＭ１２のようにＲＣ領域１０を管理あるいは制御するために固定的に必要な機能だけに限定することができる。アーキテクチャコード２０をロードしたり、マッピングする機能も、ロードするタイミングで生成したり、マッピングする領域を除いて実現することが可能であれば、ＲＣ領域１０により実現することも可能である。

【００４１】

図２にアーキテクチャコード２０を示してある。アーキテクチャコード２０は、ハードウェア回路情報２１とソフトウェア情報２２から構成される。ハードウェア回路情報２１は、ある機能単位として設計されるハードウェアモジュール（ＩＰまたはライブラリ）を１または複数に分割した分割回路１９をＲＣ領域１０の一部にマッピングする分割回路情報２３と、分割回路１９に接するインタフェース回路１８をＲＣ領域１０にマッピングするインタフェース回路情報２４とを備えている。ソフトウェア情報２２は、アーキテクチャコード２０を識別するための識別情報２５と、インタフェース回路１８において実現する境界条件２６に加え、その他の情報２７、たとえば、他の分割回路に対する優先順位、例外処理条件、動的トレード・オフ条件、分割回路の実行順序などの情報を含む。アーキテクチャコード２０により、回路を構成するプリミティブのファンクション情報やトポロジーはすべて明確にされ、個々の分割回路１９のファンクションおよび分割回路１９の接続関係も明確になる。

【００４２】

本例のデータ処理装置１においては、たとえば、Ａという機能を実現するための回路構成（ハードウェアモジュール）は、Ａ１～Ａｎのアーキテクチャコード２０として与えられる。また、Ｂという機能を実現する回路構成は、Ｂ１～Ｂｍのアーキテクチャコード２０として与えられ、Ｃという機能を実現する回路構成は、Ｃ１～Ｃｘのアーキテクチャコード２０として与えられ、Ｄという機能を実現する回路構成は、Ｄ１～Ｄｙのアーキテクチャコードとして与えられる。なお、ｎ、ｍ、ｘおよびｙは適当な整数である。

【００４３】

図３に、アーキテクチャコード２０の生成方法を示してある。まず、ステップ３１において、ハードウェアモジュールのオリジナルのネットリストを生成する。ネットリストを生成するまでの段階は、Ｃ言語などの高級言語、Ｖｅｒｉｌｏｇなどのハードウェア記述言語を用いた様々な方法が公知であり、いずれの方法を用いても良い。ステップ３２において、オリジナルのネットリストを幾つかの単位に分割し、その範囲内で、ＲＣ領域１０にマッピングできるように配置および配線問題を解決して分割回路情報２３を生成する。

【００４４】

ＲＣ領域１０にマッピングする分割回路１９は、ＲＣ領域１０を構成するハードウェアのある程度の単位（回路ブロック）を１または複数用いて配置されるように分割されている方がＲＣ領域１０に分割回路１９を効率的に割付できる。また、分割回路１９の間で頻繁にデータ交換が起きるような分割方法は、分割回路１９を個別にＲＣ領域１０にマッピングすることを考えると好ましい分割方法とは言えない。もっとも、そのような分割回路１９の組み合わせは、ＲＣ領域１０の状況が許す限り同時にマッピングするように優先順位をつけることで本発明のデータ処理装置１においてはそのような分割方法も許容するこ

とができる。いずれにしても、ステップ３２においては、オリジナルのネットリストを分割し、それをＲＣ領域１０のハードウェアに割り当てる作業をある程度繰り返して最適な分割回路情報２３が得られるようにする必要があるかもしれない。

【００４５】

さらに、ステップ３３において、オリジナルのネットリストの分割回路１９の境界を形成する情報からインタフェース回路情報２４を生成する。したがって、隣接する分割回路１９と境界が一致する部分においては、インタフェース回路情報２４は同一になり、部分的に共通のインタフェース回路情報２４を持ったアーキテクチャコードが生成されることになる。

【００４６】

次に、ステップ３４において、オリジナルのネットリストを分割回路１９の集合に変換し、それらの分割回路１９の間で、ハードウェアモジュールとしての機能が実現されるように、タイミング収束問題などを含む配置および配線問題を解決し、インタフェース回路１８における境界条件２６を生成する。したがって、隣接する分割回路１９と境界が一致または接続できる状態に対応しており、インタフェース回路情報２４が同一または対応した構成となる部分においては、境界条件も同一または対応した条件になる。このため、部分的に共通の、あるいは対応した境界条件２６を持ったアーキテクチャコードが生成されることになる。

【００４７】

さらに、分割回路を実ハードウェア空間（論理回路領域）１０にマッピングしてハードウェアモジュールとしての機能を実現させるように上記の情報をコンパイルする段階で、他の分割回路に対する優先順位、例外処理条件、動的トレード・オフ条件、分割回路の実行順序などの情報が得られるので、それらを含め、ステップ３５でアーキテクチャコード２０を生成する。したがって、ハードウェアモジュールは複数の分割回路１９により仮想ハードウェア空間上に構成され、アーキテクチャコード２０により、その一部を実ハードウェア空間であるＲＣ領域１０に実現し、実行することが可能となる。したがって、ＲＣ領域１０にマッピングされた分割回路１９は、回路インスタンスであるということができる。

【００４８】

そして、仮想ハードウェア空間と実ハードウェア空間とはインタフェース回路１９を介して結び付けられており、タイミング収束などの実ハードウェア空間に配置配線する際の問題は、インタフェース回路１９を境界条件２６により制御するという解決策が示されている。したがって、その制御をソフトウェア的に、あるいはハードウェア的に実現することが可能となる。

【００４９】

図４に、データ処理装置１において、アーキテクチャコード２０を用いてＲＣ領域１０に分割回路１９およびインタフェース回路１８を生成して実行する過程を示してある。まず、ステップ４１において、ロードユニット１３は、ＲＴＭ１２により指定されたアーキテクチャコード２０をライブラリ２から取得する。本例のデータ処理装置１においては、ＲＴＭ１２は、リスクプロセッサ６がアプリケーションプログラム４を実行する際に得られる当該データ処理装置１に対する要求、割り込み制御回路７からの割り込み情報、ＲＣ領域１０にマッピングされた分割回路１９の実行状況、ＲＣ領域１０の利用可能状況（空き領域の有無、置換可能な分割回路の有無など）を含めた動作環境情報に基づき、取得するアーキテクチャコードを決定し、ロードユニット１３に指示を出す。ロードユニット１３は、ライブラリ２にコード２０があればアドレスを出力してコード２０をフェッチする。ロードユニット１３が適当な通信機能を備えていれば、他のデータ処理装置や、外部のメモリ、さらには、ネットワークで接続されたサーバやその他のネットワーク上の資源からコード２０を取得することができる。逆に、アーキテクチャコード２０を強制的に、あるいは能動的にロードユニット１３に提供することにより、アーキテクチャコード２０を介してデータ処理装置１における処理を能動的に制御することも可能である。

【００５０】
ステップ４２において、マッピングユニット１４は取得されたアーキテクチャコード２０の分割回路情報２３およびインタフェース回路情報２４により、ＲＣ領域１０に分割回路１９と、その分割回路に接するインタフェース回路１８とをマッピングする。ＲＣ領域１０の状況は、分割回路１９の実行を制御するＲＴＭ１２が最も精度良く把握できるので、マッピングユニット１４はＲＴＭ１２の指示により、ＲＣ領域１０の空いたハードウェア空間またはリプレイス可能なハードウェア空間に分割回路１９およびインタフェース回路１８をマッピングする。その際、隣接する分割回路１９に接する境界のインタフェース回路情報２４および境界条件２６が一致あるいは対応する場合は、仮想ハードウェア空間において隣接する分割回路１９なので、実ハードウェア空間１０においてそのまま接続することが可能である。したがって、インタフェース回路を経ずに隣接する分割回路に接するように分割回路１９をマッピングする。
【００５１】
基本的には、ＲＣ領域１０の空き領域に分割回路１９はマッピングされることになる。しかしながら、ＲＣ領域１０にマッピング済みの他の分割回路に対して、ＲＴＭ１２が把握している動作環境情報によると、新たな分割回路１９を優先してマッピングする緊急性があれば、マッピング済みの他の分割回路１９を消去したり、縮小して空き領域を形成することも可能である。消去した他の分割回路１９は、緊急性が除かれた後に、ＲＣ領域１０に再度マッピングして、最初から、あるいは途中から実行することができる。また、縮小した他の分割回路１９では、分割回路１９をマッピングする工程を繰り返すことにより、処理速度は低下するが、その分割回路に係る機能の処理を継続して実行することができる。
【００５２】
ステップ４３において、マッピングされた分割回路１９を動作させる。分割回路１９を動作させるためには、ステップ４４において、境界条件２６に基づきインタフェース回路１８を制御し、分割回路１９に所定のタイミングで所定のデータを供給すれば良い。このステップ４４において、ＲＴＭ１２の機能により、ＲＣ領域１０に現在および／または過去にマッピングされた他の分割回路１９の他のインタフェース回路１８の状態が境界条件２６に基づき、動作対象の分割回路１９のインタフェース回路１８の制御に反映される。したがって、ステップ４５において、実ハードウェア空間に実現された分割回路１９は周囲に他の分割回路が接続されている仮想ハードウェア空間と同じ状態となり、ハードウェアモジュールとしての機能が実ハードウェア空間上で実現される。また、分割回路１９が動作した結果は、インタフェース回路１８に出力されるので、ＲＴＭ１２はそのインタフェース回路１８の状態をＲＣ領域１０にマッピングされている他の分割回路１９のインタフェース回路１８に空間的に伝達したり、次にマッピングされる他の分割回路１９のインタフェース回路１８に時間を経て伝達する。
【００５３】
インタフェース回路１８に設定する境界情報は、メモリ１５に記憶しておくことが可能である。他の分割回路１９がマッピングされるタイミングまでの時間が長かったり、動作途中に分割回路１９が消去されたときに、メモリ１５に記憶された境界情報をインタフェース回路１８に設定することにより、分割回路１９を所望の条件で動作、あるいは再動作させることができる。
【００５４】
ステップ４６において、マッピングされた分割回路１９を動作させる必要が終了するまでステップ４４および４５を繰り返す。そして、処理が終了した分割回路１９は、ステップ４７においてＲＣ領域１０から消去される。あるいは、ＲＣ領域１０に余裕があり、以降に分割回路１９の機能が必要となることが予測される場合は、縮小してＲＣ領域１０に存在させることも可能である。さらに、ＲＣ領域１０に余裕がある場合は、そのまま存在させておいても良い。
【００５５】

連続して入力されるデータに対して繰り返し動作が必要な分割回路１９がマッピングされている場合は、その処理が終了するまで同一の分割回路１９がＲＣ領域１０に存在する。一方、ステートマシンのように、ステートが進むことにより処理内容が順番に変わる場合は、次々に異なる分割回路１９がマッピングされる。

【００５６】

分割回路１９、すなわち、回路インスタンスは、マッピングされる際に、ハードウェア空間の動的最適化を行う為に、他の回路インスタンスの起動と消去をテーブルマスタであるＲＴＭ１２に要求することができる。ＲＴＭ１２は、複数の回路生成・消去・コピー・移動や回路間のチャネル接続を行い、本来は大規模な回路を物理空間上に展開し回路構成しないと動作しない機能を、瞬間瞬間で必要な回路だけを回路インスタンスとしてハードウェア空間に動的に最適化しながら生成し、資源の少ないハードウェア空間を用いて実質的には膨大な複数の回路を並列に動作させることができる。

【００５７】

ハードウェア空間に生成される分割回路１９は、常に、このデータ処理装置（アーキテクチャＬＳＩ）１の論理回路領域（回路プレーン）１０の上に存在するパーマネント回路と、生成された回路がある一定時間しか存在しないインスタント回路、一定時間毎に生成されるサイクリック回路等の種類に分けることができる。インスタント回路やサイクリック回路は、実際に実行されると消去される前に自分の実行結果で他の回路に通知すべき情報をＲＴＭ１２へ通知し記憶させておく。通常は、この回路実行情報は、次に生成される分割回路１９へ効率良く伝達される。逆に、ＲＴＭ１２は、インスタント回路間の実行情報が効率良く伝達されるように回路制御を行う。

【００５８】

分割回路１９の実行順序の確定は、図３に示したアーキテクチャコード２０を生成する開発段階で、開発環境（ＦＷ）の回路コンパイラがこれを行う。分割回路が、外部信号やデータ入力条件により回路実行順序に変更がある場合は、ＲＴＭ１２が、この実行制御を行う。逆に、分割回路自身で実行順番が完全に制御可能な場合は、ＲＴＭ１２がシステム全体での優先順位に応じて回路の実行エリアの拡大・縮小を行う。

【００５９】

たとえば、図１のＲＣ領域１０には、Ａ機能を実現するＡモジュールの分割回路Ａ１がインタフェース回路と共に生成され、Ｂ機能を実現するＢモジュールの分割回路Ｂ１～Ｂ３がインタフェース回路と共に生成されている。分割回路Ｂ１～Ｂ３は、連続した回路インスタンスで連続したＲＣ領域１０に生成されたので、隣接した分割回路の境界領域は連続しており、連続した分割回路から外側に繋がる境界にインタフェース回路１８が形成されている。なお、簡単に説明するためにインタフェース回路１８が図面の左右のみに生成されているが、仮想ハードウェア空間において上下に分割回路が接続される場合は、インタフェース回路が生成される場合もある。

【００６０】

Ｃ機能を実現するＣモジュールにおいては、分割回路Ｃ１およびＣ２がＲＣ領域１０にマッピングされているが空間的に分割されている。このため、各々の分割回路Ｃ１およびＣ２にインタフェース回路１８が生成され、ＲＴＭ１２を介してこれらの分割回路Ｃ１およびＣ２は接続される。また、Ｄ機能を実現するＤモジュールにおいては、分割回路Ｄ１およびＤ２が接続した状態でマッピングされている。ＲＴＭ１２は、これらの分割回路１９のインタフェース回路１８に適当なタイミングでデータをセットすることにより分割回路１８をアクティブにし、その結果インタフェース回路１８に出力されたデータを保存したり、空間あるいは時間的に分割された接続先の分割回路１９のインタフェース回路１８に伝達する。

【００６１】

さらに、ＲＴＭ１２は、分割回路１９のアーキテクチャコード２０の情報や、分割回路１９に対する動作環境情報により、ＲＣ領域１０の分割回路１９に対してクロック発生源８から供給されるクロック信号の種類、すなわち周波数を変えることができる。このため

、ＲＣ領域１０の電力消費を必要最小限に抑えることができ、パフォーマンスは最大に維持することができる。ＲＣ領域１０のうち、回路インスタンスがマッピングされていない領域にはクロック信号は原則として供給されない。

【００６２】

図５および図６は、時間が経過したＲＣ領域１０の状態である。Ａ機能はインスタント回路であり、Ａ１、Ａ２およびＡ３という分割回路１９が次々と生成されては消滅していき、その間のデータの転送はＲＴＭ１２により行われる。Ｂ機能は、図示したシーケンスでは緊急性を要する機能としてＲＴＭ１２に要求されており、ＲＣ領域１０のかなりのハードウェア資源を費やして生成されている。図５に示したタイミングでは、Ｄ機能を消滅させ、その資源を用いて多数の分割回路１９が生成されている。したがって、図６に示したタイミングでは、Ｂ機能の分割回路１９が消滅した領域にＤ機能の分割回路１９を復元して、再度、Ｄ機能の処理を途中または初めから再実行することになる。

【００６３】

図７に、ＲＣ領域１０の構成を示してある。本例のＲＣ領域１０は、各々の論理演算を変更可能な複数のエレメントの集合である回路ブロック（ｒｘｅ＿ｐｌａｎｅ）５１が格子状（アレイ状あるいはマトリクス状）に配列され、それらの間が配線５２により接続されている。アーキテクチャコード２０により定義される分割回路１９のサイズは、この回路ブロック５１の倍数の単位になっており、分割回路情報２４をコンテキスト（コンフィグレーション情報）として分割回路１９が１つまたは複数の回路ブロック５１を消費してマッピングされる。

【００６４】

図８に、１つの回路ブロック５１の構成を示してある。本例では、回路ブロック５１には、１６個の論理エレメント５３が４×４のアレイ構造をなすように配列されており、各々の論理エレメント５３は上下左右の４方向に隣接する論理エレメント５３と４ビットのレイア１のバス５４により接続されている。さらに、上下左右に隣接する論理エレメント５３を越して、その外側に位置する論理エレメント５３と接続するレイア２のバス５５も用意されており、論理エレメント５３の間のよりフレキシブルな接続が保障されている。さらに、論理エレメント５３を３つ飛び越したレイア３のバスを配置することも可能である。

【００６５】

各々の論理エレメント５３は、論理演算エレメントとしての機能と、論理エレメント間の接続切り換えを行う配線スイッチとしての機能を備えている。そして、演算する論理と、配線接続の状態を高速で変更または交換する必要があるので、本例のＲＣ領域１０には、ＲＸＥ（Rapid eXchange Element）５３と称される高速で交換動作が可能なエレメントが配置されている。

【００６６】

図９に、ＲＸＥ５３の構成を示してある。ＲＸＥ５３は、４系統の入力６１と、４系統の出力６２と、４系統の入力６１から任意の入力データを選択する入力インタフェース６３と、この入力インタフェース６３により選択された入力データφｉを論理演算して出力データを出力する演算コア６５と、４系統の入力６１と演算コア６５の出力データφｏとを任意に選択して４系統の出力６２へ接続可能な出力インタフェース６４とを備えている。演算コア６５は、論理演算を変更可能な構成になっており、論理を変更可能な演算エレメントとしての機能を果たす。また、入力インタフェース６３は、４系統の入力６１から任意の１ビットを選択するための１６対１のセレクタ６３ｓが複数配置された構成となっている。出力インタフェース６４は、演算コア６５からの出力φｏと４系統の入力６１のルーティングを兼ねた６対１のセレクタ６４ｓが複数配置された構成となっている。

【００６７】

図１０に、演算コア（ｒｘｅ＿ｃｏｒｅ）６５の構成を示してある。演算コア６５は、論理演算を指示する１６ビットのファンクションコードφｆが入力され、入力データφｉにより出力データφｏを選択するセレクタ６６と、４ビットの入力データφｉをデコード

して１６ビットのセレクタ６６の選択信号を生成するデコーダ６７と、４系統の入力６１のいずれかのデータ、または、出力データφｏをラッチするレジスタ６８と、レジスタ６８にラッチする信号を選択するためのセレクタ６９ａおよび６９ｂとを備えている。

【００６８】

図１１および図１２に、演算コア６５の動作を示している。演算コア６５はモード信号φｍによって動作が変わる。図１１のモード０では、演算コア６５は、４ビットの入力データφｉにより１ビットの出力データφｏを生成し、その出力データφｏをレジスタ６８でラッチして出力する。図１１のモード１では、演算コア６５は、４ビットの入力データφｉにより１ビットの出力データφｏを生成し、その出力データφｏをレジスタ６８でラッチせずに出力する。出力データφｏは、１６ビットのファンクションコードφｆと、入力データφｉをデコードした結果による。したがって、図１３に示すように、これらのモード１および２においては、ファンクションコードφｆを変えることにより、演算コア６５を４入力ＡＮＤから４入力コンパレータまで、９種類以上の異なる論理演算素子として使用することができる。

【００６９】

さらに、演算コア６５はセレクタ６６とファンクションコードφｆの組み合わせに論理演算を行っているので、従来のＦＰＧＡのようにＳＲＡＭなどの記憶素子を用いたルックアップテーブル（ＬＵＴ）に論理をセットする必要がない。したがって、ＳＲＡＭに入出力を行うサイクルを省略することができ、ファンクションコードφｆを演算コア６５に出力したタイミングで瞬時に演算コア６５で行う演算を交換することができる。このため、本例の演算コア６５は高速交換演算素子と称されている。

【００７０】

図１２に示したモード２からモード４においては、１つの演算コア６５が、２ビットの入力信号φｉに対して１ビットの出力信号φｏを出力する２つの演算素子として機能する。すなわち、内蔵された１６対１のセレクタ６６が、２つの４対１のセレクタとして動作するようにセットされる。これらのモード２から４においては、演算コア６５は、図１３に示してあるように、ファンクションコードφｆを変えることにより、インバータから２入力ＥＸＮＯＲまで、７種類以上の異なる論理演算素子として使用することができる。

【００７１】

さらに、図１２に示したモード５からモード７においては、演算コア６５を、３ビットの入力信号φｉに対して１ビットの出力信号φｏを出力する演算素子として使用できる。追加ビットの入力を許せば、内蔵された１６対１のセレクタ６６を、２つの３対１のセレクタとして動作するようにセットできるので、演算コア６５を２つの３ビット入力１ビット出力の演算素子としても利用できる。これらのモード５から７においては、演算コア６５は、図１３に示してあるように、ファンクションコードφｆを変えることにより、３入力ＡＮＤからフルアダーまで、５種類以上の異なる論理演算素子として使用することができる。

【００７２】

このように、本例のＲＣ領域１０を構成するＲＸＥ５３は、セレクタ方式で高速で論理を交換することが可能であり、さらに、内部に出力データをラッチするレジスタ６８を備えており、出力データをスルーで出力することも、Ｆ／Ｆによりクロックに同期した状態でも出力することができる。したがって、デジタル回路で良く使用される組み合わせ回路（デコーダ）と、順序回路（ステートマシン）及び演算回路（データパス）を、アーキテクチャコード２０の回路情報により効率よく実装し、実行することができる。

【００７３】

本例の論理を再構成可能なエレメント（ＲＸＥ）５３は、２次元アレイあるいはマトリクスを構成すること考えており、したがって、２次元に格子状に配置するのに適した４系統の入出力を備えている。しかしながら、エレメント間を接続するネットワークが１次元的であれば、２系統あるいは３系統の入出力で十分かもしれない。さらには、エレメント間を接続するネットワークが３次元的であれば、５系統以上の入出力を用意することが望

ましいであろう。さらに、本例の演算コア（ｒｘｅ＿ｃｏｒｅ）は、高速で交換動作が可能なようにセレクタ方式を採用しているが、ルックアップテーブル（ＬＵＴ）へロジックを入力するサイクルを消費できるようであればＬＵＴを備えた演算コアを採用することも可能である。また、本例においては、同一構造のエレメント５３によりマトリクスを構成しているが、論理演算用のエレメントとネットワーク形成用のエレメントによりマトリクスを構成することも可能である。さらに、算術計算を主としたエレメント、アドレス発生を主としたエレメントなどのある程度の機能に特化した、または、汎用性はあるがある機能の処理能力の高い複数種類のエレメントを適当な密度で配置したマトリクスにより、回路を再構成できるＲＣ領域を構成することも可能である。

【産業上の利用可能性】

【００７４】

システムのハードウェア・アーキテクチャは、一般的に設計開始（検討）段階で要求仕様として確定することが多い。実際のアプリケーション要求の変化や設計初期段階の時点で予想しなかった要求仕様の変更に対応する為に、最近のＦＰＧＡやＰＬＤは、ハードウェア構成を変更可能なアーキテクチャを採用している。しかし、その柔軟性自体は、内部を構成する基本エレメントを冗長化し、チップ・コスト競争力及び専用設計されたＬＳＩやＡＳＳＰに比較して動作周波数の点で不利とされ問題があった。最近は、ダイナミック・リコンフィギャブル・マシンが注目されるようになり、チップ・コストが高いという問題と動作周波数が低いという欠点をカバーするようになって来た。

【００７５】

ただ、その競争力は、１～２年掛けて開発された専用ＬＳＩと比較すると十分なレベルには無かった。本発明においては、これらの問題に加え、低消費電力化も実現することで、トータルとして現在のＳｏＣのコストパフォーマンスを実現しながら、アーキテクチャへの動的最適化を行い、次に来るハイパーＳｏＣを実現することができる。つまり、現在のＬＳＩ開発の問題点、チップ・コスト競争力と性能・低消費電力化は最高だが、開発期間と開発コストは最悪という問題を解決することができる。

【００７６】

現在のＬＳＩ設計の常識では、ハードウェア記述言語（Verilog-HDLやVHDL）を中心にして、これを各社の半導体プロセスに合うライブラリの接続形式に合うネットリストに翻訳（論理合成）する。この場合、物理配線と各論理ゲート（回路）の接続形態により動作周波数も影響を受けるが、最大の問題は、システムアーキテクチャの視点からの最適化が出来ない点にある。つまり、現在のＳｏＣやＦＰＧＡ、ダイナミック・リコンフィギャブル技術は、ハードウェア実行においてアーキテクチャ・レベルからの動的最適化の実現が出来ない構造そのものにある。

【００７７】

また、現在のＬＳＩ開発手法と実装方法では、システムの信頼性を上げたり品質を保証したりするためのコストが異常に上がる。これは、例えば、テスト回路を実装しないと内部の機能チェックが出来ないが、逆にこれを実装するとその部分のチップ面積が占められ、チップコストが上昇する。これは、結果的に品質を上げる手段は存在するが、最終的にはコストとのトレード・オフとなり信頼性や品質保証をするには限界があり、コンシュマー品に最も必要とされるテスト自体が製品の競争力を奪う結果となる。さらに、デバッグ容易化設計も全体の開発時間や開発リソースを減少する為に必要なコンセプトであるが、やはりチップ・コストを上昇させる要因となる。

【００７８】

本発明は、これらの課題の全てに対して解を与えるものである。本発明のハードウェア空間の動的最適化テクノロジーは、信頼性や品質保証する回路を必要なタイミングだけ存在させて、全体的なコスト影響を最小にできる。デバッグ容易化の為の回路は、デバッグが完了すれば一般的には必要無い。逆に、デバッグが必要なタイミングで追加すべきデバッグ用回路を生成すれば良く、本発明においては、極めて容易に対応できる。

【００７９】

さらに、アーキテクチャコードに基づく本発明は、将来、ネットワーク等を使って、動的にテスト回路やその他の機能を実現する回路を変更したり生成したりすることを可能とし、大規模で複雑なシステムを構築するコストを大幅に低減できる。したがって、手元には小型のチップ化された本発明のデータ処理装置が内蔵されたターミナルを持ち、ネットワークを介して膨大なリソースを持つ仮想ハードウェア空間と接続することにより、多種多様な機能を手元の小型のターミナルにより実行することが可能となる。このシステムは、ネットワークを介して膨大な入出力データを通信しながらネットワーク上に存在するハードウェア資源を用いて処理を行う現在の方式とは全く逆の発想であり、ネットワーク上に存在するハードウェア資源を手元のターミナルで実行しようというものである。したがって、大量の入出力データの送受信を緩和してネットワーク負荷を低減でき、また、データの秘匿性を保証できるなど、様々なメリットを持ったシステムが本発明に基づき構築されうる。

【００８０】

また、上記においては、半導体集積回路技術をベースにしたＬＳＩに本発明を適用する例を説明しているが、いわゆる回路網を形成するデータ処理装置のすべてに本発明を適用することが可能である。すなわち、電気あるいは電子レベルの回路技術をベースにしたデータ処理装置に限らず、光、生体、分子あるいは原子構造、遺伝子構造などをベースにした回路網を形成する全てのデータ処理装置に対して本発明を適用することができる。

【図面の簡単な説明】

【００８１】

【図１】本発明のデータ処理装置の概略構成を示す図である。

【図２】アーキテクチャコードの概要を示す図である。

【図３】アーキテクチャコードを生成する過程を示すフローチャートである。

【図４】データ処理装置においてアーキテクチャコードを実行する過程を示すフローチャートである。

【図５】ＲＣ領域の構成の一例を示す図である。

【図６】ＲＣ領域の構成の異なる例を示す図である。

【図７】ＲＣ領域のハードウェア構成を示す図である。

【図８】エレメントの配置を示す図である。

【図９】エレメントの構成を示す図である。

【図１０】演算コアの構成を示す図である。

【図１１】演算コアの動作例を示す図である。

【図１２】演算コアの他の動作例を示す図である。

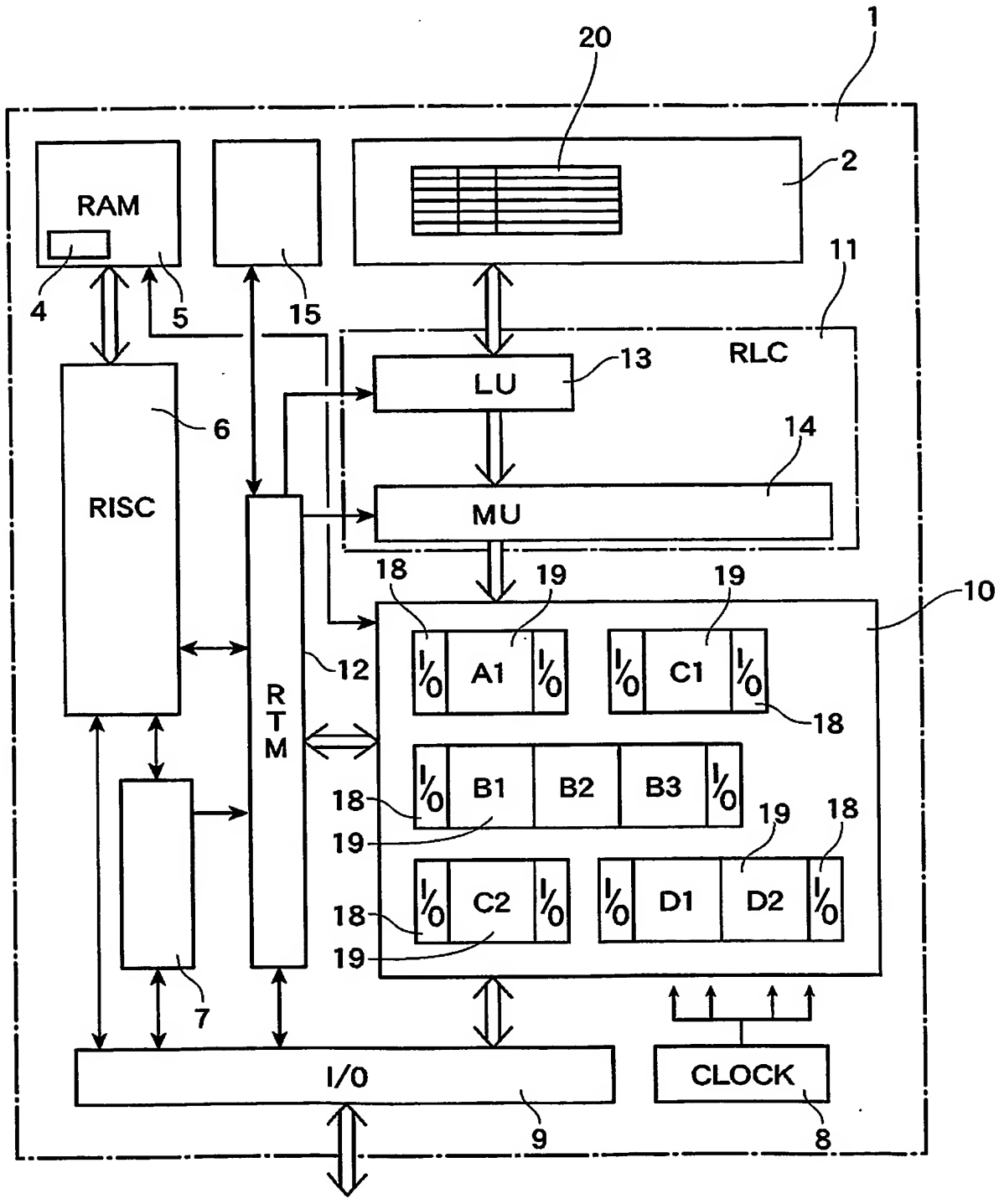
【図１３】演算コアで実行可能な論理演算の例を示す図である。

【符号の説明】

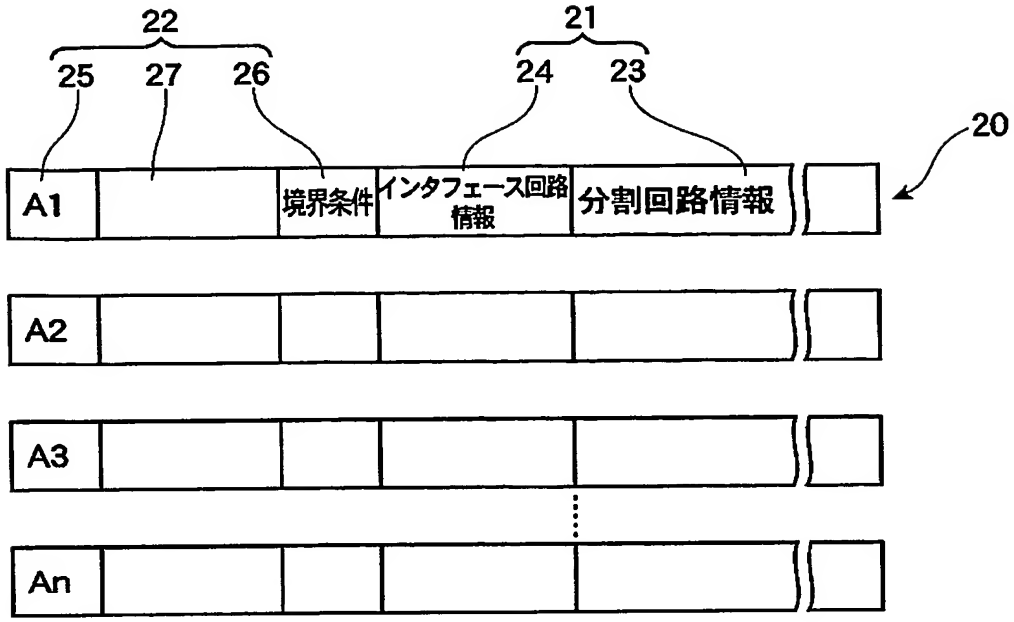
【００８２】

１　データ処理装置
１０　再構成可能な論理回路領域（ＲＣ領域）
１１　高速ローディング制御ユニット（ＲＬＣ）
１２　高速論理通信マスタ（ＲＴＭ）
１３　ロードユニット
１４　マッピングユニット
５１　回路ブロック（ｒｘｅ＿ｐｌａｎｅ）
５３　論理演算素子（ｒｘｅ）
６５　演算コア（ｒｘｅ＿ｃｏｒｅ）

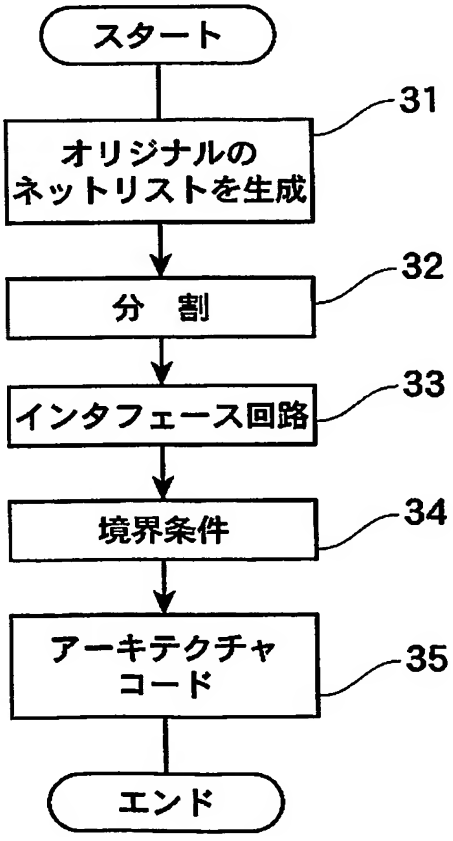
【書類名】図面
【図１】


1
20
2
RAM
4
5
15
11
RLC
13
LU
6
14
RISC
MU
18
19
19
10
12
I/O
A1
C1
RTM
18
B1
B2
B3
18
19
19
18
C2
D1
D2
18
19
7
I/O
CLOCK
9
8

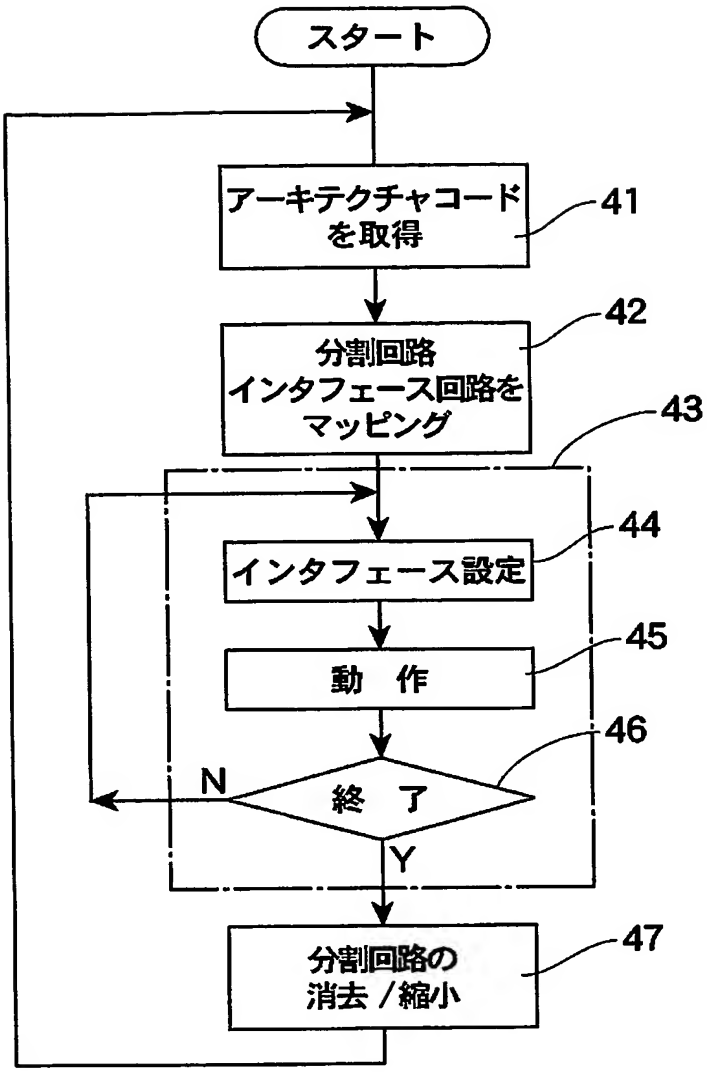
【図２】


22
21
25
27
26
24
23
20
A1
境界条件
インタフェース回路情報
分割回路情報
A2
A3
An

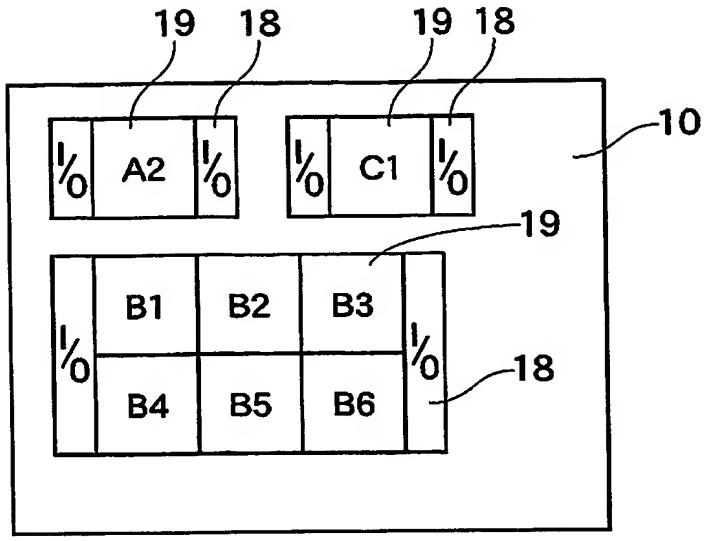
【図３】


スタート
31
オリジナルのネットリストを生成
32
分　割
33
インタフェース回路
34
境界条件
35
アーキテクチャコード
エンド

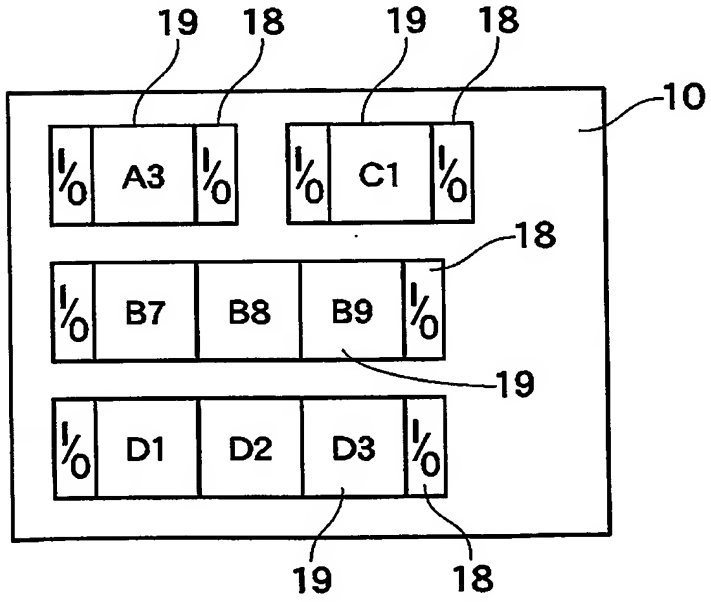
【図４】


スタート
アーキテクチャコードを取得
41
分割回路インタフェース回路をマッピング
42
43
インタフェース設定
44
動　作
45
終　了
46
N
Y
分割回路の消去／縮小
47

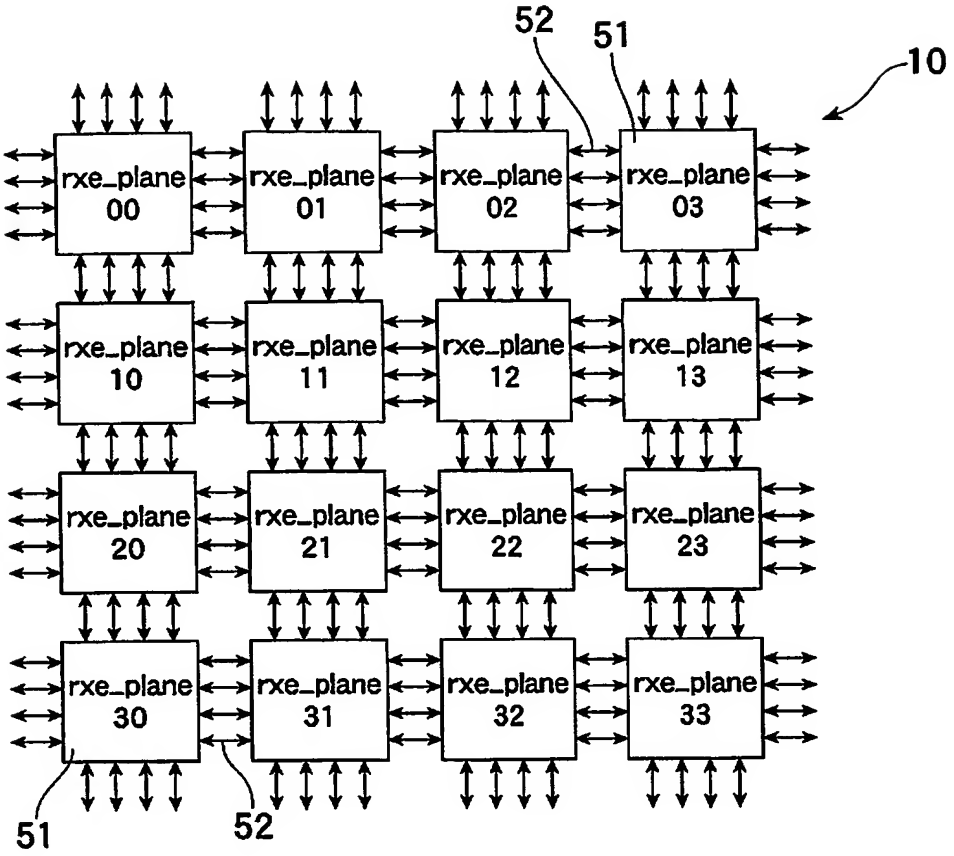
【図５】


19
18
19
18
10
I/O
A2
I/O
I/O
C1
I/O
19
B1
B2
B3
I/O
I/O
18
B4
B5
B6

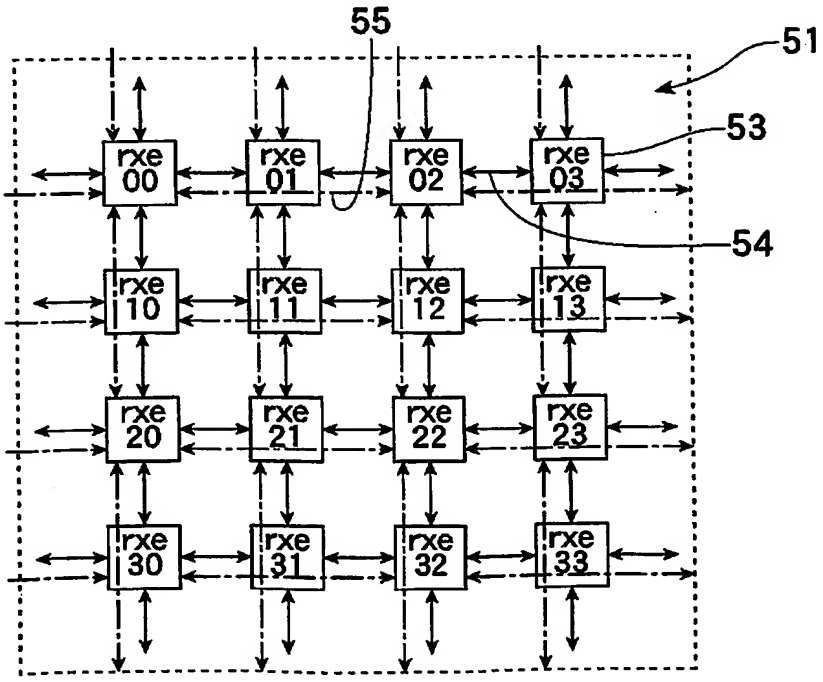
【図６】


19
18
19
18
10
I/O
A3
I/O
I/O
C1
I/O
18
I/O
B7
B8
B9
I/O
19
I/O
D1
D2
D3
I/O
19
18

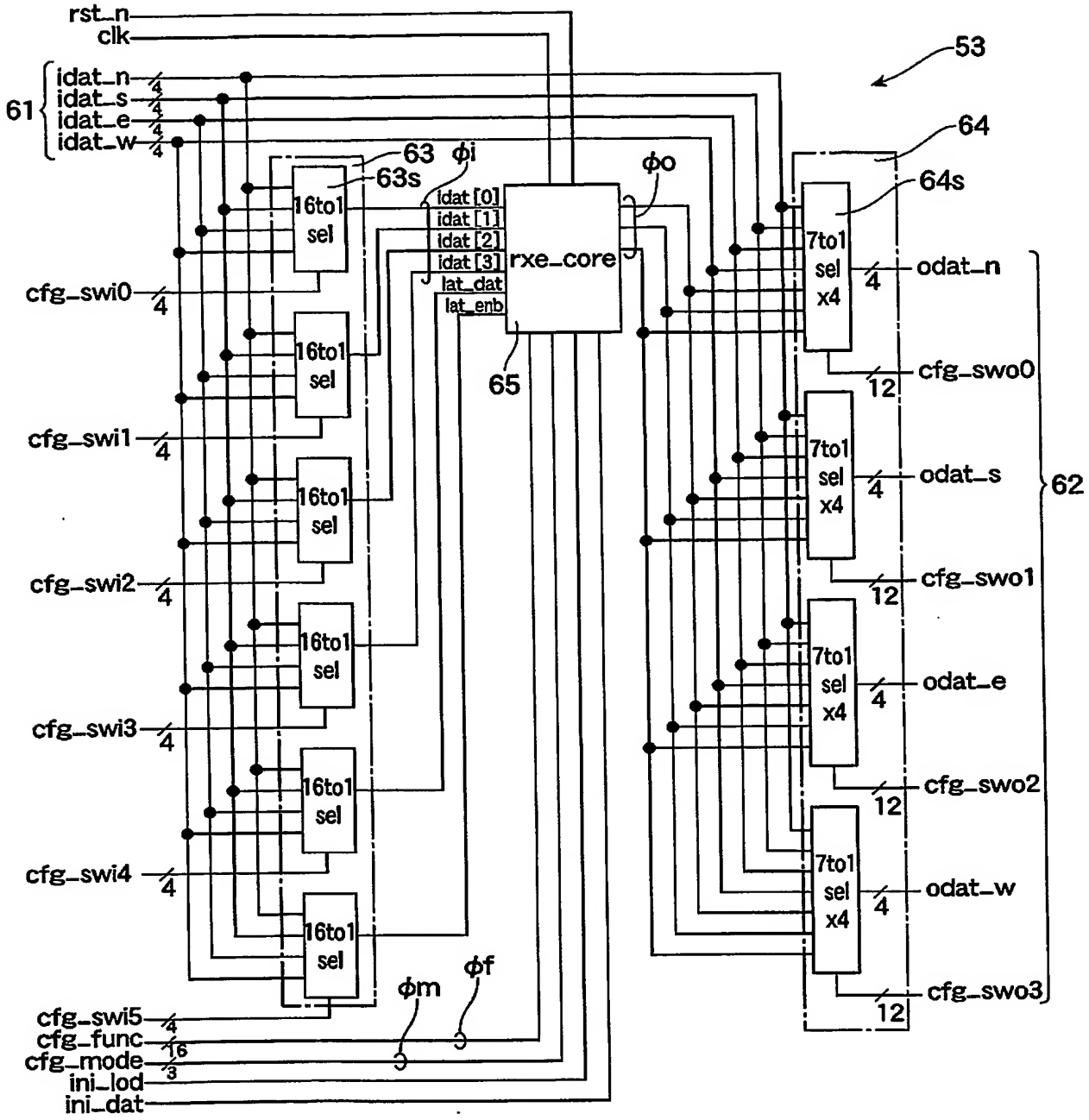
【図７】


52
51
10
rxe_plane 00
rxe_plane 01
rxe_plane 02
rxe_plane 03
rxe_plane 10
rxe_plane 11
rxe_plane 12
rxe_plane 13
rxe_plane 20
rxe_plane 21
rxe_plane 22
rxe_plane 23
rxe_plane 30
rxe_plane 31
rxe_plane 32
rxe_plane 33
51
52

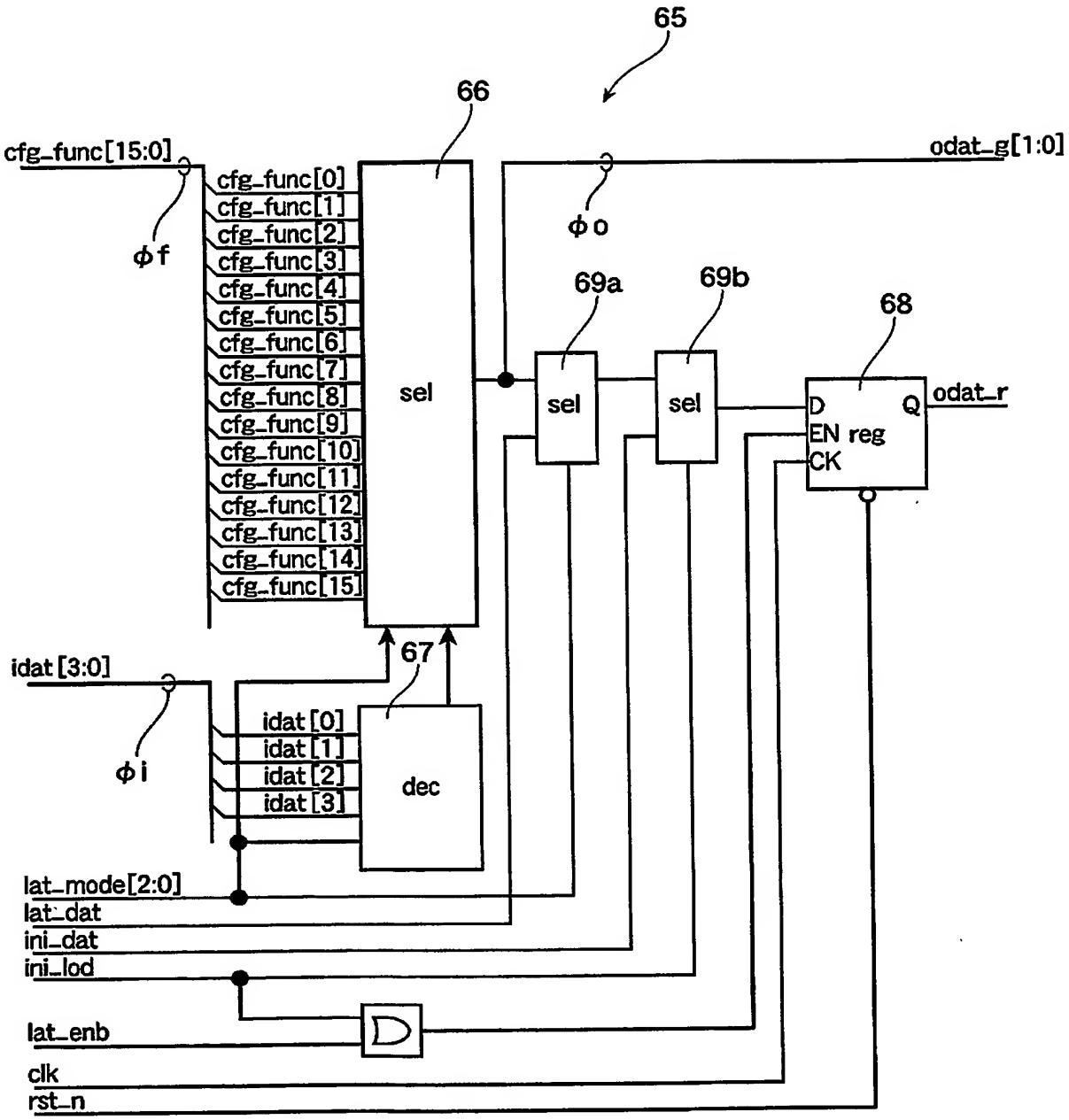
【図８】


55
51
53
54
rxe 00
rxe 01
rxe 02
rxe 03
rxe 10
rxe 11
rxe 12
rxe 13
rxe 20
rxe 21
rxe 22
rxe 23
rxe 30
rxe 31
rxe 32
rxe 33

【図９】


rst_n
clk
61
idat_n
idat_s
idat_e
idat_w
53
64
64s
63
63s
φi
φo
16to1 sel
idat [0]
idat [1]
idat [2]
idat [3]
lat_dat
lat_enb
rxe_core
65
7to1 sel x4
odat_n
odat_s
odat_e
odat_w
cfg_swo0
cfg_swo1
cfg_swo2
cfg_swo3
62
cfg_swi0
cfg_swi1
cfg_swi2
cfg_swi3
cfg_swi4
cfg_swi5
cfg_func
cfg_mode
ini_lod
ini_dat
φm
φf

【図１０】


65
66
cfg_func[15:0]
cfg_func[0]
cfg_func[1]
cfg_func[2]
cfg_func[3]
cfg_func[4]
cfg_func[5]
cfg_func[6]
cfg_func[7]
cfg_func[8]
cfg_func[9]
cfg_func[10]
cfg_func[11]
cfg_func[12]
cfg_func[13]
cfg_func[14]
cfg_func[15]
φf
sel
φo
odat_g[1:0]
69a
69b
68
sel
sel
D
Q
EN reg
CK
odat_r
idat[3:0]
idat[0]
idat[1]
idat[2]
idat[3]
φi
67
dec
lat_mode[2:0]
lat_dat
ini_dat
ini_lod
lat_enb
clk
rst_n

【図１１】

	φm		φi		φf		φo	
cfg-mod [2:0]	idat [3]	[2]	[1]	[0]	odat_g [1]	[0]	odat_r	備考
000 (4in1out) [モード0]	0	0	0	0	0	cfg_func [0]	cfg-func [0]	odat-g [0]の値を保持
	0	0	0	1	0	cfg_func [1]	cfg-func [1]	
	0	0	1	0	0	cfg_func [2]	cfg-func [2]	
	0	0	1	1	0	cfg_func [3]	cfg-func [3]	
	0	1	0	0	0	cfg_func [4]	cfg-func [4]	
	0	1	0	1	0	cfg_func [5]	cfg-func [5]	
	0	1	1	0	0	cfg_func [6]	cfg-func [6]	
	0	1	1	1	0	cfg_func [7]	cfg_func [7]	
	1	0	0	0	0	cfg_func [8]	cfg_func [8]	
	1	0	0	1	0	cfg_func [9]	cfg_func [9]	
	1	0	1	0	0	cfg_func [10]	cfg_func [10]	
	1	0	1	1	0	cfg_func [11]	cfg_func [11]	
	1	1	0	0	0	cfg_func [12]	cfg_func [12]	
	1	1	0	1	0	cfg_func [13]	cfg_func [13]	
	1	1	1	0	0	cfg_func [14]	cfg_func [14]	
	1	1	1	1	0	cfg_func [15]	cfg_func [15]	
001 (4in1out) [モード1]	0	0	0	0	0	cfg_func [0]	lat_dat	レジスタ分離
	0	0	0	1	0	cfg_func [1]	lat_dat	
	0	0	1	0	0	cfg_func [2]	lat_dat	
	0	0	1	1	0	cfg_func [3]	lat_dat	
	0	1	0	0	0	cfg_func [4]	lat_dat	
	0	1	0	1	0	cfg_func [5]	lat_dat	
	0	1	1	0	0	cfg_func [6]	lat_dat	
	0	1	1	1	0	cfg_func [7]	lat_dat	
	1	0	0	0	0	cfg_func [8]	lat_dat	
	1	0	0	1	0	cfg_func [9]	lat_dat	
	1	0	1	0	0	cfg_func [10]	lat_dat	
	1	0	1	1	0	cfg_func [11]	lat_dat	
	1	1	0	0	0	cfg_func [12]	lat_dat	
	1	1	0	1	0	cfg_func [13]	lat_dat	
	1	1	1	0	0	cfg_func [14]	lat_dat	
	1	1	1	1	0	cfg_func [15]	lat_dat	

【図１２】

cfg-mod [2:0]	idat [3]	[2]	[1]	[0]	odat_g [1]	[0]	odat-r	備考
010 (2in1out) [モード2]			0	0		cfg_func [0]	cfg_func [0]	上位2ビットと下位2ビットは独立した2系統になるレジスタは下位2ビットの結果を保持
			0	1		cfg_func [1]	cfg_func [1]	
			1	0		cfg_func [2]	cfg_func [2]	
			1	1		cfg_func [3]	cfg_func [3]	
	0	0			cfg_func [8]			
	0	1			cfg_func [9]			
	1	0			cfg_func [10]			
	1	1			cfg_func [11]			
011 (2in1out) [モード3]			0	0		cfg_func [0]		上位2ビットと下位2ビットは独立した2系統になるレジスタは上位2ビットの結果を保持
			0	1		cfg_func [1]		
			1	0		cfg_func [2]		
			1	1		cfg_func [3]		
	0	0			cfg_func [8]		cfg_func [8]	
	0	1			cfg_func [9]		cfg_func [9]	
	1	0			cfg_func [10]		cfg_func [10]	
	1	1			cfg_func [11]		cfg_func [11]	
100 (2in1out) [モード4]			0	0		cfg_func [0]	lat_dat	上位2ビットと下位2ビットは独立した2系統になるレジスタ分離
			0	1		cfg_func [1]	lat_dat	
			1	0		cfg_func [2]	lat_dat	
			1	1		cfg_func [3]	lat_dat	
	0	0			cfg_func [8]		lat_dat	
	0	1			cfg_func [9]		lat_dat	
	1	0			cfg_func [10]		lat_dat	
	1	1			cfg_func [11]		lat_dat	
101 (3in1out) [モード5]	x	0	0	0	cfg_func [8]	cfg_func [0]	cfg_func [0]	MSBは未使用 odat-g[0]の値を保持
		0	0	1	cfg_func [9]	cfg_func [1]	cfg_func [1]	
		0	1	0	cfg_func [10]	cfg_func [2]	cfg_func [2]	
		0	1	1	cfg_func [11]	cfg_func [3]	cfg_func [3]	
		1	0	0	cfg_func [12]	cfg_func [4]	cfg_func [4]	
		1	0	1	cfg_func [13]	cfg_func [5]	cfg_func [5]	
		1	1	0	cfg_func [14]	cfg_func [6]	cfg_func [6]	
		1	1	1	cfg_func [15]	cfg_func [7]	cfg_func [7]	
110 (3in1out) [モード6]	x	0	0	0	cfg_func [8]	cfg=func [0]	cfg_func [8]	MSBは未使用 odat-g[1]の値を保持
		0	0	1	cfg_func [9]	cfg_func [1]	cfg_func [9]	
		0	1	0	cfg_func [10]	cfg_func [2]	cfg_func [10]	
		0	1	1	cfg_func [11]	cfg_func [3]	cfg_func [11]	
		1	0	0	cfg_func [12]	cfg_func [4]	cfg_func [12]	
		1	0	1	cfg_func [13]	cfg_func [5]	cfg_func [13]	
		1	1	0	cfg_func [14]	cfg_func [6]	cfg_func [14]	
		1	1	1	cfg_func [15]	cfg_func [7]	cfg_func [15]	
111 (3in1out) [モード7]	x	0	0	0	cfg_func [8]	cfg_func [0]	lat_dat	MSBは未使用 レジスタ分離
		0	0	1	cfg_func [9]	cfg_func [1]	lat_dat	
		0	1	0	cfg_func [10]	cfg_func [2]	lat_dat	
		0	1	1	cfg_func [11]	cfg_func [3]	lat_dat	
		1	0	0	cfg_func [12]	cfg_func [4]	lat_dat	
		1	0	1	cfg_func [13]	cfg_func [5]	lat_dat	
		1	1	0	cfg_func [14]	cfg_func [6]	lat_dat	
		1	1	1	cfg_func [15]	cfg_func [7]	lat_dat	

【図１３】

ファンクション	cfg-mode [2:0]	cfg-func [15:0]	備考
インバータ	010/011/100	xxxx_xxxx_xxxx_0101	最下位ビットを使用
2入力 AND	010/011/100	xxxx_xxxx_xxxx_1000	下位2ビットを使用
2入力 NAND	010/011/100	xxxx_xxxx_xxxx_0111	下位2ビットを使用
2入力 OR	010/011/100	xxxx_xxxx_xxxx_1110	下位2ビットを使用
2入力 NOR	010/011/100	xxxx_xxxx_xxxx_0001	下位2ビットを使用
2入力 EXOR	010/011/100	xxxx_xxxx_xxxx_0110	下位2ビットを使用
2入力 EXNOR	010/011/100	xxxx_xxxx_xxxx_1001	下位2ビットを使用
3入力 AND	101/110/111	xxxx_xxxx_1000_0000	下位3ビットを使用
3入力 NAND	101/110/111	xxxx_xxxx_0111_1111	下位3ビットを使用
3入力 OR	101/110/111	xxxx_xxxx_1111_1110	下位3ビットを使用
3入力 NOR	101/110/111	xxxx_xxxx_0000_0001	下位3ビットを使用
FullAdder	101/110/111	1110_1000_1001_0110	下位3ビットを使用 上位出力にキャリー 下位出力にサム
4入力 AND	000/001	1000_0000_0000_0000	
4入力 NAND	000/001	0111_1111_1111_1111	
4入力 OR	000/001	1111_1111_1111_1110	
4入力 NOR	000/001	0000_0000_0000_0001	
4入力 EXOR	000/001	0111_1111_1111_1110	
4入力 NOR	000/001	1000_0000_0000_0001	
AND_AND_OR	000/001	1000_1000_1000_1000	
AND_AND_NOR	000/001	0111_0111_0111_0111	
4入力 comparater(1111)	000/001	1000_0000_0000_0000	比較したい値を1にする

【書類名】要約書
【要約】
【課題】　回路を動的に再構成可能な実ハードウェア空間を動的に最適化することができるデータ処理装置を提供する。
【解決手段】　回路を動的に再構成可能なＲＣ領域１０と、アーキテクチャコード２０を取得するロードユニット１３と、アーキテクチャコード２０により、ＲＣ領域に分割回路１９とインタフェース回路１８とをマッピングするマッピングユニット１４と、アーキテクチャコード２０にしたがってインタフェース回路を制御するＲＴＭ１２とを有するデータ処理装置１を提供する。アーキテクチャコード２０は、ある機能単位として設計されるハードウェアモジュールを１または複数に分割した分割回路１９をＲＣ領域１０の一部にマッピングする分割回路情報と、分割回路１９に接するインタフェース回路１８をＲＣ領域１０にマッピングするインタフェース回路情報と、インタフェース回路において実現する境界条件とを備えている。
【選択図】　図１

認定・付加情報

特許出願の番号	特願２００３－３０６３５７
受付番号	５０３０１４３５１０１
書類名	特許願
担当官	第八担当上席　　００９７
作成日	平成１５年　９月　１日

＜認定情報・付加情報＞

【提出日】　平成15年 8月29日

特願２００３－３０６３５７

出願人履歴情報

識別番号　［５００２３８７８９］

１．変更年月日　２００３年　６月　２日
［変更理由］　住所変更
住　所　東京都品川区上大崎二丁目２７番１号
氏　名　アイピーフレックス株式会社